Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 16 (2013) 118 – 127

Procedia Computer Science

Conference on Systems Engineering Research (CSER'13)
Eds.: C.J.J. Paredis, C. Bishop, D. Bodner, Georgia Institute of Technology, Atlanta, GA, March 19-22, 2013.

Integrating analytical models with descriptive system models: implementation of the OMG SyML standard for the tool-specific case of MapleSim and MagicDraw

Sebastian J. I. Herzig[a]*, Nicolas F. Rouquette[b], Stephen Forrest[b,c], J. Steven Jenkins[b]

[a]*Georgia Institute of Technology, Atlanta, Georgia, United States of America*
[b]*Jet Propulsion Laboratory, Pasadena, California, United States of America*
[c]*Maplesoft, Waterloo, ON, Canada*

Abstract

The Jet Propulsion Laboratory (JPL) is investing heavily in the development of an infrastructure for building system models using the Systems Modeling Language (SysML). An essential component is a transformation apparatus that permits diverse models to be integrated independently of their nature (e.g. declarative, analytical and statistical). This paper presents one useful case: the integration of analytical models expressed using the Modelica language. Modelica is an open standard, declarative, multi-domain modeling language that allows for complex dynamic systems to be modeled. Maplesoft's MapleSim is one software tool that supports the Modelica language. The tool-neutral specification for the transformation between the languages Modelica and SysML is defined in the SysML-Modelica transformation specification (SyML) standard published by the Object Management Group (OMG). As part of the development efforts, said specification has been implemented using the Query-View-Transformation Operational (QVTO) language. During the process, several critical changes to the current SyML standard were proposed. Furthermore, a number of current limitations related to MapleSim were identified. Despite these issues, a proof-of-concept transformation was successfully implemented. In conclusion, the integration of complex simulation models conforming to the Modelica language with SysML-based system models has shown great promise and is a highly useful tool to support the decision making process in design.

Keywords: model transformation; modelica; sysml; model-based systems engineering; model integration

1. Introduction

A great number of systems, especially those found in the aerospace industry, all share a common property: they are highly complex. In fact, most are so complex, that it is nearly impossible for a single human being to keep track

* Corresponding author. Tel.: +1-404-247-0290
E-mail address: sebastian.herzig@gatech.edu

doi:10.1016/j.procs.2013.01.013

of all the specifics. Yet, decisions need to be made to push development progress forward while, at the same time, making sure that no inconsistencies are present in the information and knowledge contributed by the various stakeholders, of which each may have a unique view on the same system [1]. All of this knowledge and information must come together to make informed decisions. To manage this inherent complexity, a number of methods were developed and *Systems Engineering* as a discipline was formed [2, 3].

Traditionally, consistency management and information interchange is enabled by a document-based approach: stakeholders document their work, while systems engineers make sure that the results meet the requirements and the system specification remains consistent and coherent at all times [3]. The latter is, for the most part, done by cross-checking documents manually. Such methods are prone to human error and can lead to difficulties in terms of maintaining consistency over the course of development. This is primarily due to many of the relations and dependencies between the different documents being implicit.

In this day and age, where access to computational infrastructures is highly affordable, one would think that relations between such documents can be defined more explicitly. To a certain degree, tools such as PLM systems can aid in tracing information. However, one great challenge that still remains is that capturing information and knowledge in documents is informal and, without additional information, cannot be interpreted unambiguously. Therefore, it is valid to ask: how might one improve on the document-based approach? One approach towards avoiding ambiguities and introducing a formal way of documenting the work products of different stakeholders is *Model-Based Systems Engineering* (MBSE).

Model-Based Systems Engineering is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases [3]. Storing information associated with a given system in a formal, computer-interpretable and model-based manner, allows for relations such as property dependencies to be expressed explicitly. The main artifact associated with MBSE is a system model, which contains a descriptive system specification, design, analysis and verification information. Its primary purpose is to enable the design of a system that satisfies its requirements and supports the allocation of requirements to the system's components [3]. One modeling language that allows for system models to be expressed is the Systems Modeling Language (SysML) [4], a standard maintained by the Object Management Group (OMG).

One step towards a model-based development environment is an apparatus that allows for models to be related to one another. Within the context of model-centric development, this is known as a transformation apparatus [5]. Model transformations should allow for diverse models to be integrated, independent of their nature (e.g. stochastic, descriptive, statistical and analytical). This paper investigates the integration of a specific type of analysis models with a system model. As part of the work, the OMG SysML-Modelica transformation standard SyML [6] was implemented for the tool-specific case of Maplesoft's Modelica-supporting tool MapleSim. The resulting model transformation apparatus was integrated into JPL's NoMagic MagicDraw UML-based modeling infrastructure. The implementation and the investigated approach of separating a model transformation from the tools involved are the primary contributions of this paper. Furthermore, a number of proposed enhancements to the SyML standard are presented.

Integrating analytical models with descriptive system models is highly important in that it enables a system model to include decision rationale. Furthermore, the integration is advantageous, in that the descriptive part of the system model can be used to automatically generate parts of an analysis model. Such a capability can help reduce the non-value added work associated with creating the initial structure and interface of an analysis model and hence also reduce cost and risk. Another reason that makes the importance of the integration apparent is the potential for using sophisticated analyses for automated detection of inconsistencies through execution of test cases that verify requirements within the system model.

2. Background & Related Work

In principle, there are three ways in which information (and knowledge) contained in engineering tools can be exchanged with other stakeholders: the first, and currently most commonly used approach, is the use of documents. This has already been identified as an informal approach. The second method makes use of tool-capabilities such as an *Application Programming Interface* (API). Such an approach works well in cases where all of the tools involved

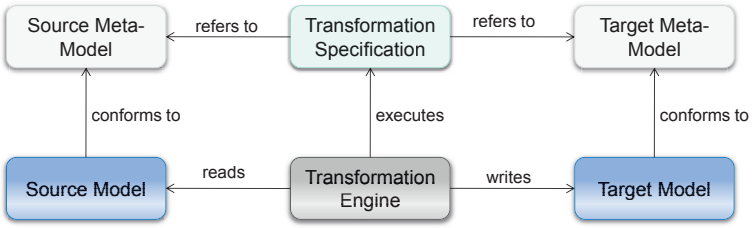

Fig. 1. Concept of model transformations: transforming a source model into a target model representation [5]

are available on the same computer and only local changes are made. However, in a collaborative environment, a third approach known as *model integration* is a more valuable option.

The basic principle of model integration is that one works with a model produced in a given engineering tool rather than with the tool itself. Models conform to a modeling language (e.g. Modelica [7] or SysML) and can be related to other models by means of a model transformation. *Transformation specifications* contain the rules of the corresponding transformation and are commonly based on the definition of the modeling languages involved in the process [5]. For example: a modeling language such as UML or SysML defines a concept of *properties*, which may have values. Modelica offers a similar capability through its concept of *variables*. In a model transformation, such properties and variables can be related to one another. A transformation engine uses the transformation specification to translate a source model to a target model. This process is depicted in Figure 1. The interested reader is encouraged to read more about model transformations in the related literature.

Model integrations have a variety of advantages over tool integrations: firstly, it is a more "natural" integration of information and knowledge since the underlying syntax and semantics of the models is used rather than a tool interface. This makes it a similar approach to using documents, in which not the tools, but information and knowledge related to what was modeled in a tool is communicated. Separating models from tools has an additional benefit: different tools supporting the same modeling language can, at least theoretically, be used interchangeably, while not having to rewrite the complete transformation for a specific tool.

Numerous research and development efforts have investigated the use of model transformations between different kinds of models. Johnson et al. discuss the integration of models and simulations of continuous dynamics into SysML, mentioning Modelica as a key enabler [8]. Within the context of integrating specifically Modelica-conformant models with descriptive models, the work of Schamai et al. [9] should be mentioned. The authors developed an integration of the Modelica language and UML. Work by Pop et al. resulted in an implementation of an integrated environment in Eclipse [10]. Similarly, Reichwein developed an Eclipse-based environment that integrates numerous modeling languages, one of which is Modelica [11]. These and select other efforts lead to the development of the SyML standard for transforming between SysML and Modelica conforming models. This standard was later adopted by the OMG [6]. At JPL, numerous ongoing and previous efforts have resulted in tool-based and model-based integrations into the previously mentioned modeling infrastructure.

3. Implementation of the Model Integration using the SyML Standard

In the previous section it was determined that a formal way of integrating model-based information and knowledge is by model integration. By doing so, the information and knowledge captured in the individual models is related rather than the corresponding tools. The SyML standard builds on this idea by providing a transformation specification between the two modeling languages OMG SysML and Modelica 3.1 [7]. In other words, as long as a pair of SysML and Modelica models are conformant to their respective language definitions (irrespective of the tools they were created in), an associated model transformation can be made completely independent from any supporting tool. This is the key idea behind the approach taken in the presented work. More details regarding the specific workflow, especially related to the transition from an analytical to a descriptive perspective as well as aspects of consistency management, are given in the following sub-sections.

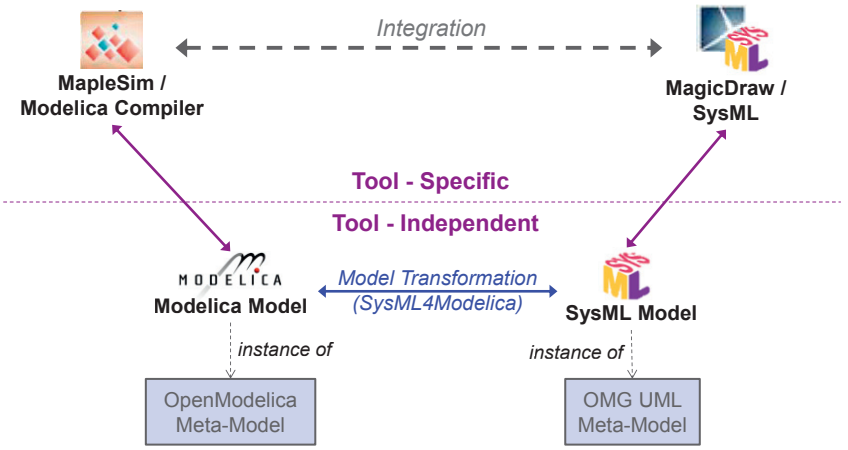

Fig. 2. Overview of approach taken: notice that the approach has been subdivided into a tool-dependent and a tool-independent part

3.1. Overall Approach for the Integration into JPL's Modeling Infrastructure

Transformation specifications can be implemented in a variety of ways. One method is to utilize a general purpose programming language such as Java to interpret the source model and generate (or update / synchronize) a target model. These operations are typically based on the meta-model of the language and on knowledge about the structural semantics. While this approach seems fairly attractive from an ease-of-implementation perspective, it also has its drawbacks: in most instances of developing model integrations, the meta-model of at least one of the models involved is dynamic in nature, i.e., the meta-model may change over time. Accounting for this temporal evolution of a meta-model is possible, but cumbersome: it may be necessary to change large portions of the algorithmic logic during the evolution of the meta-model. Maintenance of such an algorithmically defined model transformation can incur high costs related to adapting to changes in corresponding language definitions. Hence, a better alternative is needed. One option is to use a language designed specifically for the purpose of defining a model transformations. Such languages are called *transformation languages* and allow for the definition of a mapping between two languages to be specified explicitly. OMG's *Query-View-Transformation* (QVT) standard [12] is the basis for one such language: QVT Operational (QVTO).

JPL's modeling infrastructure includes a QVTO interpreter that is based on the open-source Eclipse QVTO transformation engine and is directly integrated with NoMagic's MagicDraw. Several enhancements and modifications to the original version have been made. These include a caching mechanism and broader support of the QVTO language [13]. For this reason and due to the previously stated advantages over other general purpose programming languages, QVTO has been chosen as the language to implement the transformation specification.

The given QVTO transformation execution engine expects both the source and target model to conform to Ecore-based meta-models. NoMagic's proprietary UML meta-model or one native to the Eclipse Modeling Framework can be used for the purposes of representing the SysML model. For Modelica models, there are a variety of Ecore-based meta-models available. One example is the openModelica meta-model, published and maintained by the *Open Source Modelica Consortium* (OSMC) [14]. This meta-model has become a part of the OMG SyML standard.

Given that the source and target models conform to an Ecore-based meta-model, the QVTO-based transformation specification can be implemented independent from the tools used to author the corresponding models in. As depicted in Figure 2, the transformation should, ideally, take place between an openModelica-conformant and an OMG UML-conformant model. The only tool-specific part that remains is the generation of the respective source model and the interpretation of the target model. Following such an approach reflects the previously proposed idea of splitting the transformation into a tool-independent and a tool-specific part.

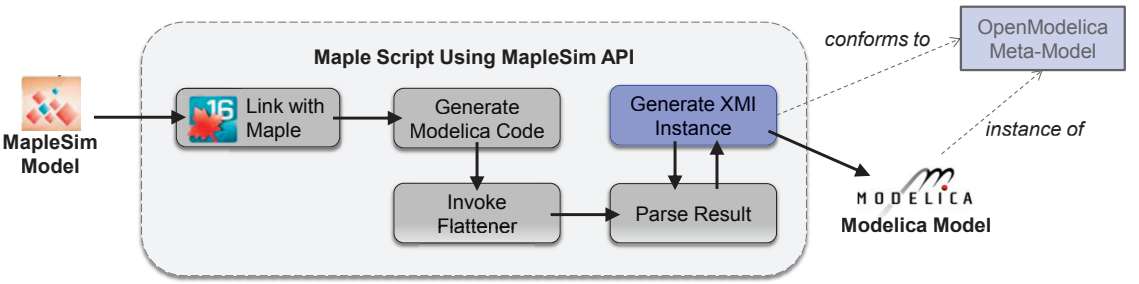

Fig. 3. Generation of the openModelica conformant model from a given MapleSim model

3.2. Tool-Specific Model Generation & Interpretation

In recent versions, MagicDraw has started support for Ecore-based meta-models. JPL's modeling infrastructure has taken advantage of this by integrating the QVTO transformation execution engine in such a way, that the model loaded in MagicDraw can directly be accessed during the execution of a transformation specification. *Blackbox libraries* and a set of operations in QVTO (or, in QVTO-speak, *helper functions*, *queries* and *mappings*) have been developed as part of the modeling infrastructure to ease the manipulation of UML-conforming models. These libraries include a set of functions that allow one to apply stereotypes to model constructs, given that the corresponding profile is loaded within MagicDraw. Hence, the tool-specific portion of the model integration that requires interpretation and manipulation of UML-conforming models within MagicDraw is already available.

Unfortunately, at the time of writing this paper, MapleSim does not offer any support for Ecore-based models. Hence, a capability of making the information contained within MapleSim available to the QVTO transformation execution engine had to be developed. In the same spirit, an interpreter that provides MapleSim with the necessary information to either construct or update an existing model in its internal representation was conceptualized. These procedures are, of course, highly tool-dependent.

When developing this tool-dependent part of the model integration, it was taken into account that Ecore-based models can be stored as *XML Metadata Interchange* (XMI) [15] conformant files. Hence, given that a Modelica model is available in some form – be it a proprietary, internal format or a format conforming to some standard – one can parse the available data and reconstruct an openModelica-conformant model simply by constructing its XMI representation. However, making the information required to construct the openModelica-conformant representation available through MapleSim is not a trivial task.

Several alternative ways to extract the information associated with a given MapleSim model were analyzed. One conceptualized alternative makes use of the fact that the graphical user interface of MapleSim is written in Java. It was attempted to retrieve the required information by manually invoking the graphical user interface and, on the same Java virtual machine, accessing the dynamically created objects that internally represent the currently loaded model. However, this alternative was quickly dismissed, since some of details required to construct the openModelica-conforming model were not used in the graphical front-end and, hence, were never stored in any of the dynamic objects. After consulting the help of Maplesoft, it was discovered that most of the logic associated with compiling and executing Modelica models was contained in a library of routines written in the programming language native to Maple. The routines and the associated documentation are commonly hidden from users and only a limited set of functions representing an API is exposed. Also, much of the information previously not accessible (e.g. inheritance relationships of library components), is available in a similar form, external to the graphical user interface. Using the documentation provided by Maplesoft, it was possible to construct the openModelica-conformant model by parsing Maple's internal representation of the Modelica model. As depicted in Figure 3, this process involves the linking of a MapleSim model with a Maple worksheet, and invoking a preprocessor on the corresponding Modelica code. The preprocessor returns a structure corresponding to a directed acyclic graph, which is then parsed and the XMI representation constructed. A similar procedure was conceptualized for the interpretation of an existing openModelica-conformant model: re-construct the MapleSim-internal representation of the Modelica

model by parsing a given XMI representation of a Modelica model.

A third alternative was also investigated, which was based on using an existing Modelica parser that is capable of interpreting the Modelica modeling language in its textual form and outputting a model conforming to the openModelica meta-model. One such parser is part of the openModelica compiler suite [14]. However, this alternative was not pursued further, mainly due to the fact that MapleSim relies in part on an extended, proprietary set of library components that the openModelica parser would not be able to resolve. Also, the alternative was dismissed due to the parser not being able to take the changes that were made to the openModelica meta-model into account. These changes are discussed in the next section.

3.3. Changes to the openModelica Meta-Model

A number of changes were made to the available version of the openModelica meta-model. One rather critical modification was the redesign of the mechanism allowing for the referencing of externally defined Modelica model elements (e.g. declaration of types by referencing components from the Modelica standard library). Another change consisted of the refactoring of the model by removing all sub-packages and setting the parent of all Ecore class to the root package. Furthermore, all abstract classes that have only one concrete implementation were removed and type specifications updated accordingly.

The publicly available openModelica meta-model allows for externally defined Modelica language constructs to be referenced using one of two methods: the first involves specifying a qualified path to an element as a dot delimited string. The second method entails the construction of a qualified path by using a nested structure of meta-classes named *uPath*. uPath has three concrete specializations: FULLYQUALIFIED, QUALIFIED and IDENT. To construct a qualified path, several instances of the QUALIFIED meta-class are nested to represent the path, with the reference of the last element pointing to an instance of IDENT. For example, to construct the qualified path "MyLibrary.MySubPackage.MyElement", three classes are created: a class of meta-type QUALIFIED referring to "MyLibrary" references another instance of QUALIFIED representing "MySubPackage". This last instance references an instance of meta-class IDENT, representing "MyElement". The difference between QUALIFIED and FULLYQUALIFIED is the base reference point: while QUALIFIED is used to specify a qualified path relative to a given element, FULLYQUALIFIED specifies a qualified path relative to the model root. This method of referencing elements works well for the case where it is assumed that all model elements are defined relative to a single root element. Such is commonly the case when working within a Modelica tool environment. However, as will become evident in the next section, this referencing mechanism does not suffice for unambiguously retaining references to externally defined model elements during a model transformation. Details of the newly introduced referencing mechanism are provided thereafter.

3.4. Model Transformation

As mentioned previously, the model transformation is designed to be independent from any specific SysML- or Modelica-modeling tool. Several activities are executed as part of the transformation process: first, references to external model elements are resolved. Then, a subset of the elements of the Modelica model is selected according to a set of rules. In a third step, the resulting partial representation of the complete Modelica model is transformed into a SysML-conformant representation using the rules specified in the OMG SyML standard. Conformance to the SyML standard includes the application of stereotypes from the UML profile that is available as part of the standard. All of these steps are carried out in a transformation specification written in QVTO. The remainder of this subsection is concerned with the second and third stage of the transformation process. Establishing references is related to the newly implemented referencing mechanism and is covered in more detail in the section that follows.

During the second stage, a subset of the model elements that make up the Modelica model is selected. This subset represents the descriptive representation of the analysis. The decision of not using all model elements is based on the belief that not all details of an analysis model must necessarily be contained in the system model. After all, the intent is not to execute an analysis within the system model, but to establish associations between components of the system model and an analysis model. Furthermore, transforming all of the information will not only clutter the model, thereby making it less readable, but also complicate the task of making sure that both representations are consistent at all times. Therefore, not all of the information and knowledge contained within the Modelica model

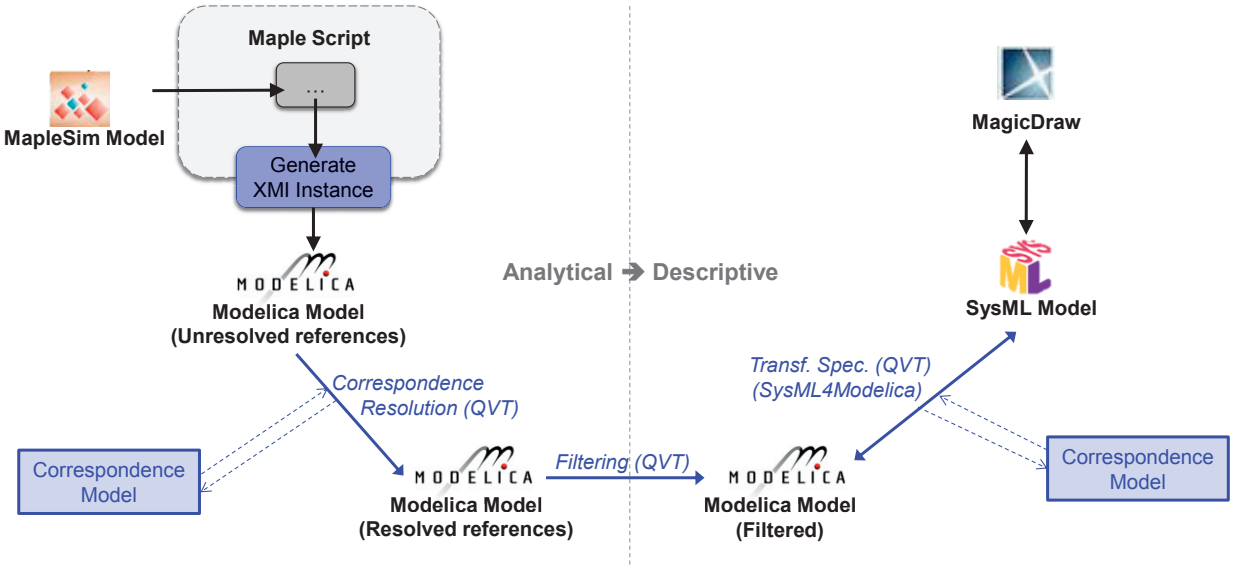

Fig. 4. Model generation, correspondence resolution and translation from an analytical representation to a descriptive one

should be transformed. The subset of the information selected for transformation is equivalent to the descriptive representation of the analysis model, i.e. this step represents the transition from an analytical to a descriptive representation of a model.

The third step consists of the process of transforming the selected subset of information into a SysML-conformant representation. It should be noted that this transformation step is intended to be executed bi-directionally: in other words, all of the information that was selected to be part of the descriptive part of the Modelica-conformant analysis model is transformed into an equivalent SysML-conformant representation and any changes in the SysML representation are synchronized. The implementation of this stage of the transformation is based on the QVTO code distributed with the SyML standard. However, most of the implementation was re-written and large parts of it were extended. Some parts were believed to be non-conformant to the SyML standard and were re-implemented. This includes the handling of class inheritance.

3.5. Ensuring Consistency: Defining Correspondences between Model Constructs

As mentioned in section 3.3, the mechanism for referencing elements outside the local scope of a given Modelica model (e.g. referencing types defined in the Modelica standard library) has been redesigned. The governing principle of the new method is relatively simple: instead of using qualified paths, elements are referenced by a globally unique element property. In Ecore, this translates to a simple reference (EReference) in one model to an element contained within an externally defined model, with both models conforming to the same meta-model. Hence, given that a dependent model exists and can be loaded dynamically, a reference to it can be established. For example, consider a Modelica model that contains a class that owns a property *myProp* of type *C*. Furthermore, say that *C* is a Modelica class declared in the Modelica standard library and is contained within a package called *P*. Instead of using the fully qualified path "*P.C*" to resolve the type, a reference to an openModelica-conformant model containing the declaration of *P.C* is established. The aforementioned process is executed during the first stage of the transformation process by dynamically loading all available models (e.g. an openModelica-conformant representation of the Modelica standard library), querying for the sought-after type and referencing elements accordingly.

While the aforementioned works well for establishing unique references to elements among different Modelica models, at least one other challenge related to consistency management still remains: preserving the correspondence between elements defined in a Modelica model and the corresponding SysML-conformant representation. However,

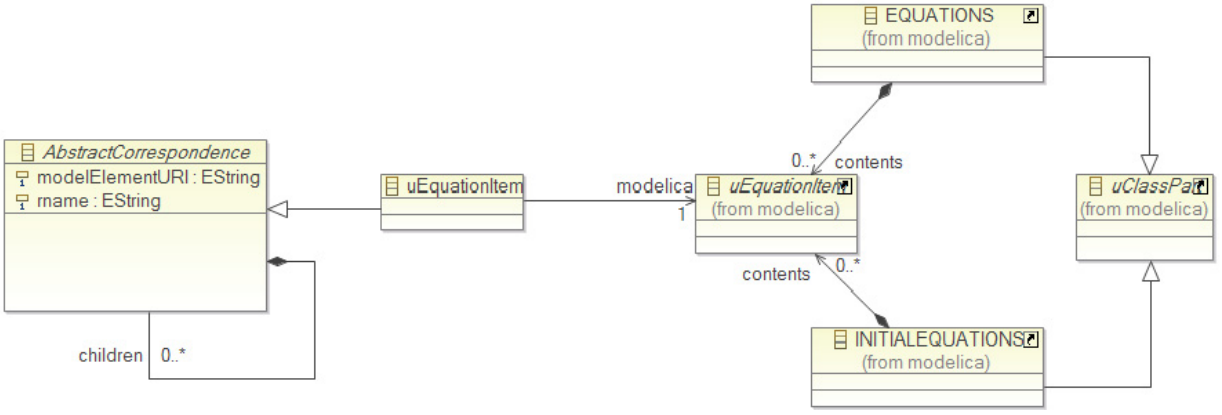

Fig. 5. Extract of the correspondence meta-model: notice that elements in the SysML model are referenced by a unique URI, which is generated by MagicDraw

given that information about the correspondence is available, references across different Modelica models represented in a SysML model can also be inferred from associations established within the SysML-conformant representations: in other words, if two or more *canned* (partial representation of the Modelica model), SysML-conform representations of previously transformed analysis models are related to one another - for instance, by interconnecting their respective input and output parameters – references can, at least theoretically, be inferred. This correspondence is an important consideration to be made, since model elements may be located at different depths and locations relative to the given model root in the corresponding Modelica- or SysML-conformant representations. Hence, qualified paths can also no longer be used to unambiguously resolve types across both representations.

So, how does one establish correspondence relationships between the same model elements across two different representations? Again, the basis for tackling this challenge is formed by making use of globally unique identifiers of model elements. However, in this case, elements that are part of two distinct models that each conform to a different modeling language must be related to one another. As part of the solution approach, a meta-model was conceptualized that allows one to reference elements from both the openModelica meta-model and NoMagic's proprietary UML meta-model (see figure 5). During the model transformation, a single instance of this meta-model is handled and updated accordingly. Elements that have previously not been transformed are added, while modifications to elements existing in both representations are propagated through the established references. In essence, this *correspondence model* has a similar functionality to that of a database.

4. Results

During the course of the development of the integration between MapleSim and JPL's modeling infrastructure, several challenges had to be tackled. Some of these lead to recommendations for changes to existing standards. Others resulted in suggestions for tool enhancements. For example, while Modelica 3.1 supports multiple class inheritance, the SyML standard does not. In terms of suggestions for tool improvements, Maplesoft was recommended to extend their publicly available MapleSim API. All in all, valuable insight into the integration of models of various natures has been gained, especially the integration of analytical models and system models.

The departure from the idea of simply transforming all of the information contained within an analysis model into a descriptive model had the important result of being able to differentiate between an analytical and a descriptive representation of one and the same analysis model. In the end, this proved to be a valuable idea, since redundant and, to a certain degree, unnecessary information would have otherwise cluttered the system model. Furthermore, the idea of using canned representations of analysis models to compose them into more complex analysis models within the system model has arisen. This is investigated in currently ongoing work.

Preliminary results of this work include a partially complete implementation of the SyML transformation standard and its integration into JPL's modeling infrastructure. The primary purpose of the implementation is to

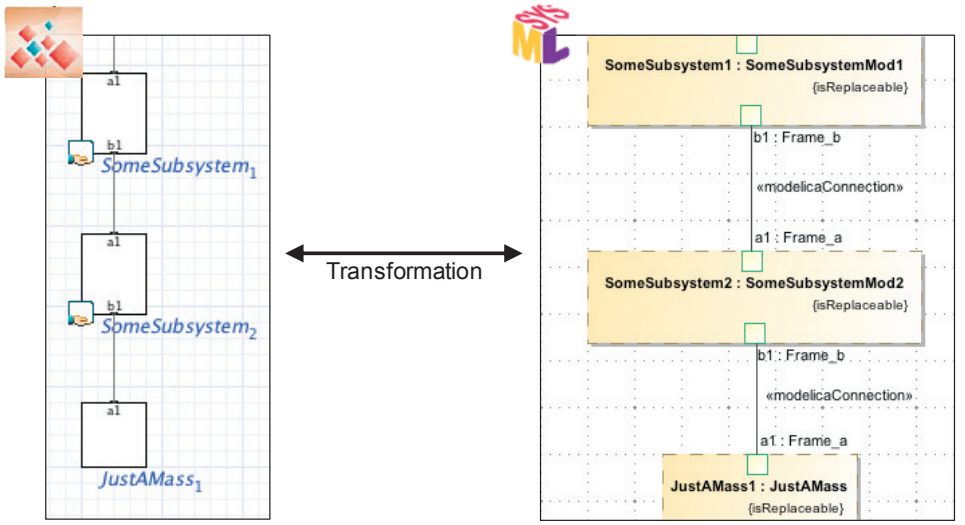

Fig. 6. Double pendulum example: Modelica model created in MapleSim (left) and corresponding representation in SysML (right)

serve as a demonstrator and motivator for future work. Several examples and use case scenarios were created to complement the integration. One such example is a double pendulum (see figure 6), which was created in MapleSim (left) and then transformed to SysML (right).

5. Conclusions & Future Work

The work presented in this paper consists of the integration of one particular class of models – analytical models - with system models expressed using SysML. More specifically, analysis models conforming to the Modelica 3.1 language were integrated into JPL's existing modeling infrastructure. This existing model-centric development infrastructure enables the construction of SysML-conformant system models using NoMagic's MagicDraw UML tool and allows for model integration by means of an integrated, enhanced version of the Eclipse QVTO transformation execution engine. Maplesoft's MapleSim was selected as the Modelica modeling tool to be used.

Within the context of this work, the key idea behind the model integration is to split the model transformation process into a tool-specific and a tool-independent part. The tool-independent part is a self-contained model transformation based on the existing OMG SyML standard and the tool-specific part requires a programmable interface to the modeling tools. It should be noted that such an approach requires knowledge about the modeling languages supported by each tool.

From the discussion provided, it should be apparent that the integration of analytical models with descriptive system models can be very powerful. Partly, this is due to the emerging possibility of automatically generating parts of the structure of an analysis model from a descriptive representation of a system (i.e. the system model). Given that Modelica is an object-oriented modeling language, it is even feasible to capture decision rationale at various levels of fidelity by refining a single general analysis by means of specialization. An additional use case for the proposed integration is the application of the results of a given analysis to refine and verify existing requirements, thereby avoiding internal inconsistencies [1].

Overall, the model integration of analytical models with descriptive models is an important next step towards a model-centric development of a system. In the scope of this work, only a partial, proof-of-concept implementation was completed. However, the authors believe that there is enough evidence to justify subsequent work. Therefore, the authors recommend further investigation and the completion of the integration.

Acknowledgements

The principal author would like to express his gratitude towards the JPL Education Office for the financial support, without which the work would have never been possible. Also, the authors would like to thank Maplesoft for the more than exemplary technical support. Furthermore, the input from Dr. Christiaan Paredis of the Georgia Institute of Technology, Dr. Axel Reichwein of Koneksys LLC and Alek Kerzhner of the Jet Propulsion Laboratory is greatly appreciated.

References

1. S. J. I. Herzig, A. Qamar, A. Reichwein, and C. J. J. Paredis, "A Conceptual Framework for Consistency Management in Model-Based Systems Engineering," *ASME Conference Proceedings*, pp. 1329–1339, 2011.
2. "INCOSE Systems Engineering Handbook," 2000.
3. S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML*. The MK/OMG Press, 2008.
4. "SysML, The Systems Modeling Language (OMG SysML(TM)) Version 1.2," June 2010.
5. K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches," in *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.
6. "SysML-Modelica Transformation (SyM)," June 2012.
7. Modelica Association, "The Modelica Language Specification Version 3.1," 2010.
8. T. Johnson, A. Kerzhner, C. J. J. Paredis, and R. Burkhart, "Integrating Models and Simulations of Continuous Dynamics Into SysML," *Journal of Computing and Information Science in Engineering*, vol. 12, no. 1.
9. W. Schamai, P. Fritzson, C. J. J. Paredis, and A. Pop, "Towards Unified System Modeling and Simulation with ModelicaML: Modeling of Executable Behavior Using Graphical Notations," in *Proceedings 7th Modelica Conference*, 2009.
10. A. Pop, D. Akhvlediani, and P. Fritzson, "Integrated UML and Modelica System Modeling with ModelicaML in Eclipse," in *Proceedings of the 11th IASTED International Conference on Software Engineering and Applications*, SEA '07, (Anaheim, CA, USA), pp. 557–563, ACTA Press, 2007.
11. A. Reichwein, *Application-Specific UML Profiles for Multidisciplinary Product Data Integration*. PhD thesis, 2012.
12. Object Management Group (OMG), "Meta Object Facility (MOF) 2.0 Query/View/Transformation, V1.1," 2011.
13. N. Rouquette, "JPL Eclipse QVTO Patch Overview," tech. rep., Jet Propulsion Laboratory (JPL), 2012.
14. "Open Source Modelica Consortium (OSMC).", http://www.openmodelica.org/
15. Object Management Group (OMG), "MOF/XMI Specification Version 2.4.1," 2011.